Teaching and learning with computers

Stefano Penge, may 2022

Part I

Tools in education

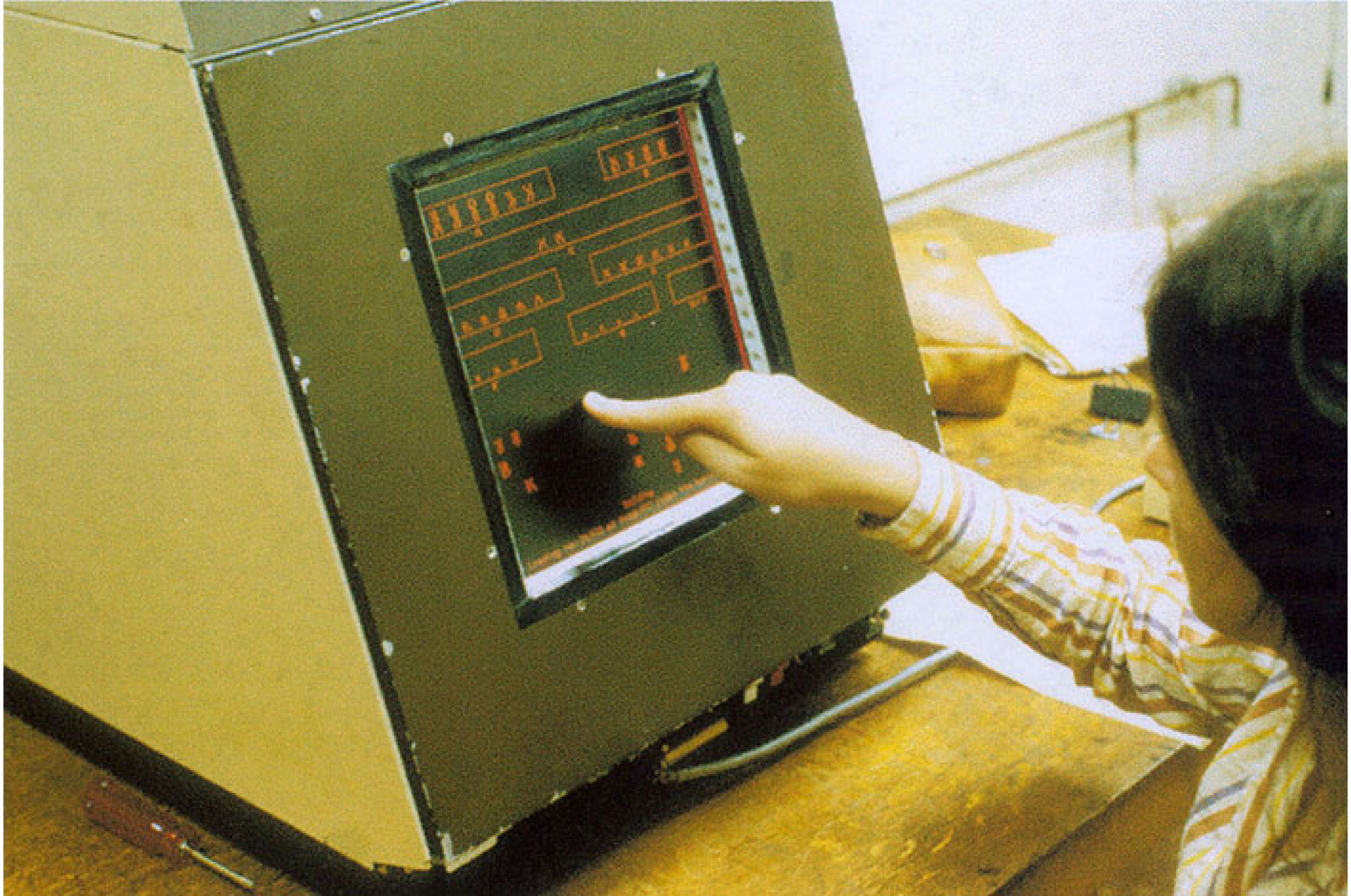
- The idea of a "age of gold" of education in which there were a direct relationship between teacher and pupils, without mediation, is a false myth
- First of all, teachers are mediators themselves
- Second, there is always a mediation: voice, posture, position in the space
- Moreover, there are (always) been tools to help and support teaching

Simple Tools

- Textbooks (read only)
- Blackboards (w/r)
- Copybooks (w/r)
- Maps (read only?)
- Dictionaries (read only?)
- ...

Complex tools

- gramophones and discs
- audio players and recorders
- radio and tv
- computers
- internet
- mobile digital tools
- wearable tools
- …?



CAI

- Computer Aided Instruction
- From the very beginning computers were used to substitute teachers in assessment, in giving interactive lessons with media
- The **PLATO** project (Programmed Logic for Automatic Teaching Operations, 1960-2006, University of Illinois and CDC company) was probably the first time computer were used systematically to teach and learn
- Behind it there was a behaviouristic theory of learning : learning is just reacting correctly to a class of stimuli.
- A learning path can be designed in advance for *every* subject matter
- But students are different and the path should be adapted to everyone
- Here comes the role of Artificial Intelligence

Intelligent Tutoring Systems /1

- ITS are one of the first “applications” of AI (Student, Daniel Bobrow 1964) but also the model upon which the idea of Cognitive AI was built
- The Lisp Tutor (1983), Parnassus (1989), Auto TUTOR (1999)
- In this vision:
  - It is possible to design an explicit model of concepts and relations in a domain (eg. algebra, computer programming)
  - It is possible to build a model of the subset of that domain that lives in the student's mind
  - It is possible to assess the state of this model with quizzes and after this update it is possible to rearrange subsequent steps

Intelligent Tutoring Systems /2

- Limits:
  - It is ok
    - with a) adults b) motivated
    - in closed and formalized domains
  - It is based on a single pedagogical model, a single domain model, a single student model
- Still nowadays, it stands as a reference for using AI and ICT in education (see Socratic) or as a polemical target (the end of teachers)

Part II
Educational software

Educational software

- Nowadays there is a lot of educational software on/ and offline
- We can divide ES in two main categories:
  - full functioning environments (https://phet.colorado.edu/it/)
  - open environments
- The former are easy to deal with, but are little customizable or not at all
- The latter are highly customizable, but they are difficult to understand, at least for absolute beginners

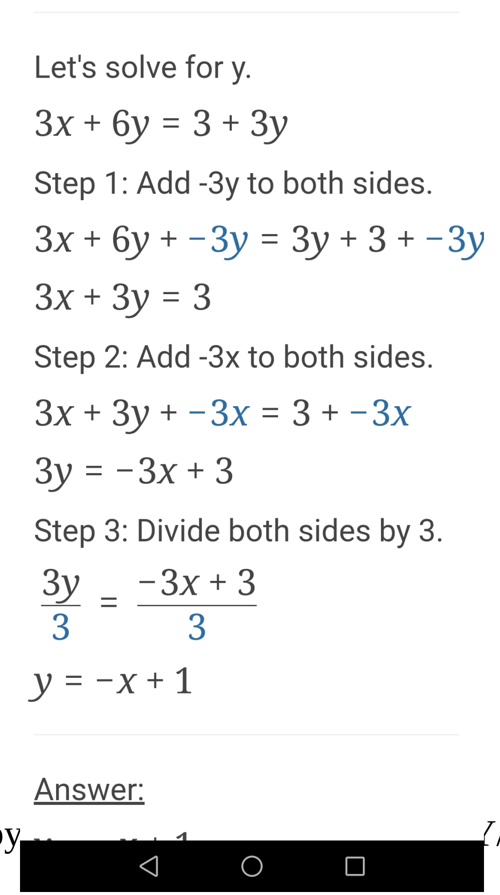
An example: Socratic

- Socratic was born as an app/website where teachers gave support to pupils.
- Then it became a standalone app based on opensource libraries to transform an image in formal representation of an equation and to solve it showing the steps
- Socratic aim to be a tutor:
  - An intelligent tutor accompanying the student and show how to do, gives suggestions
  - A tutor that never got tired, available at any moment, in every place, not only in classrooms and at school time.
- As many other "intelligent" services (search, maps, translation) its final goal is to become indispensable
- https://youtu.be/vZ1tQZ8glXg

Socratic

$3x+6y = 3 + 3y$

- Cymath
- Mathpapa
- Wolfram Alpha



Let's solve for y.

$3x + 6y = 3 + 3y$

Step 1: Add -3y to both sides.

$3x + 6y + -3y = 3y + 3 + -3y$

$3x + 3y = 3$

Step 2: Add -3x to both sides.

$3x + 3y + -3x = 3 + -3x$

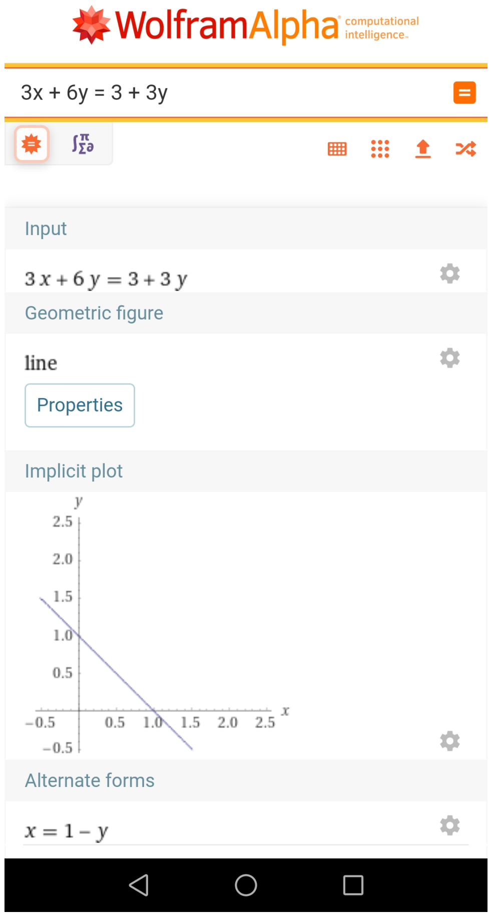
$3y = -3x + 3$

Step 3: Divide both sides by 3.

$$\frac{3y}{3} = \frac{-3x + 3}{3}$$

$y = -x + 1$

Answer:



WolframAlpha computational intelligence.

3x + 6y = 3 + 3y

Input

$3x + 6y = 3 + 3y$

Geometric figure

line

Properties

Implicit plot

Alternate forms

$x = 1 - y$

Educational languages 1

- Since the very beginning, it was clear that to have the number of "coders" needed, it was necessary to imagine languages and environments much more simpler that those used by the first programmers
- COBOL (1961 COmmon Business-Oriented Language) was the first language aimed not to engineers but to "normal" people
- BASIC (Beginners' All-purpose Symbolic Instruction Code, 1964) was also aimed to allow everyone to program without being an engineer
- Pascal (1970) was invented to help young programmers to learn structured programming and to avoid the nightmare of *spaghetti programming*
- On one side there were "real" programming languages (like C), powerful, concise (1 instruction = 1 word), with complex and flexible syntax
- On the other side there were languages more verbose (1 instruction = many words), with terms easy to remember, with simple and rigid syntax
- "Real programmers don't use Pascal", Ed Post, https://www.ee.ryerson.ca/~elf/hack/realmen.html

Educational languages 2

- A few years later, researchers like Seymour Papert and others started to think of computer as good environments not only to learn programming, but to learn everything *while* programming
- Programming was thought about as having a talk with computers
- Reversing the classical teaching situation, programming was seen as way *to teach* to computers (define a function = teach the meaning of a new word)
- Logo language (1967-) was explicitly designed with this approach in mind, to learn geometry but also to "play with powerful ideas"
- But there were also Alice, Etoy, Squeak, Scratch, Snap!, ToonTalk, Kodu, Kojo, ...

The ideas behind

- To teach a language it is important to know:
  - the language itself
  - the history (why it was designed in that way? which problems did it solve? which were the models and pitfalls to avoid?)
  - the limits
  - the competitors, the alternatives

Logo by Brian Harvey

- *One difference between a Logo microworld and an ordinary piece of educational software is that in the Logo version, the program itself is available for inspection and modification, not a black box. In the study of computer science, a programming language is itself a microworld.*
- *A crucial part of a Logo classroom is that the learners have permission to explore using their own individual styles. There isn't just one right way to program*
- *The irony in all this, as we'd expect from our Logo experience, is that the students who end up doing best, even in the job market, are the ones who can find the courage to forget about jobs and grades and jumping through hoops*
- https://people.eecs.berkeley.edu/~bh/elogo.html

Scratch by Mitchell Resnick

- *One problem was that Logo, like most programming languages, required children to learn obscure punctuation rules, such as where to put colons and square brackets. This distracted from the most important ideas and goals of programming.*
- *We wanted Scratch to engage young people from many different backgrounds and in many different contexts, including homes, schools, community centers, libraries, and museums.*
- *We wanted the community to serve as a source of inspiration and feedback: young people could see what others were creating (and borrow pieces of the code for their own projects),*
- *[…] we had seen that many young people prefer to learn new skills and techniques by exploring examples rather than following a lesson or tutorial.*
- https://medium.com/@mres/10-sparks-that-lit-the-flame-of-scratch-595a27d44334

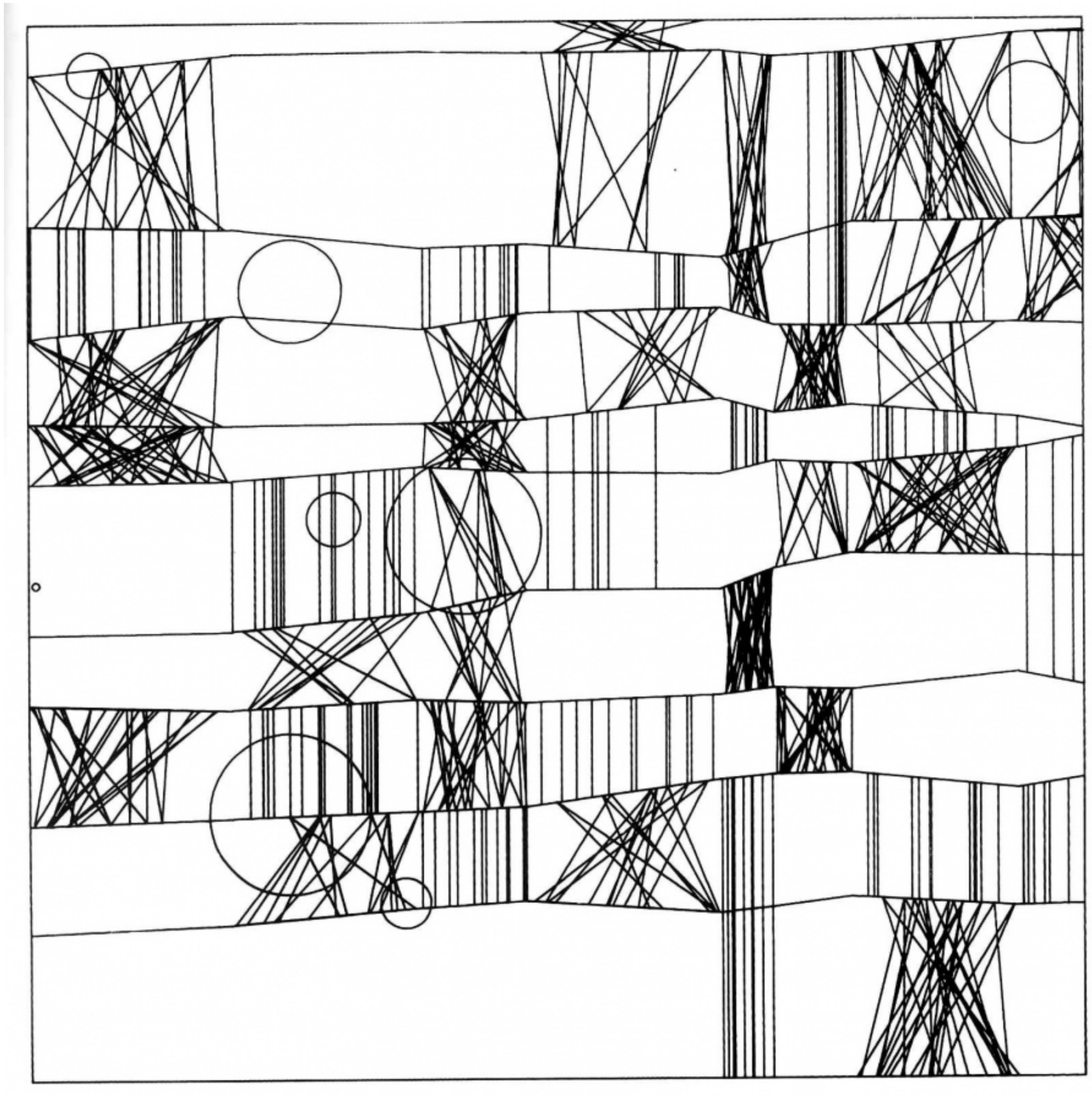
Problems

- We often find a definition of programming as "implementing an algorithm to solve a problem"
- This is a simplified vision of programming, as a functional/ practical activity. But there is much more in programming
- There are programs the do no solve anything but create pieces of art (music, image, animation, text)
- There are languages invented just for fun: they are the so called "Esoteric languages"
- Problems arise, and solving problems is a typical developer's task; but it is not *the aim of programming* in itself

Digital art

- Art an computers did cross many times in the past
  - Software as painters (Nake, 1960)
  - Software as poets (Balestrini, 1961)
  - Software as musicians (Gross, 1970: TAUMUS https://www.codeshow.it/Codici/TAUmus)

Frieder Nake,
1960 ?
Probably with a
Zuse plotter



AI artists

- Xiaoice (2017) is a Microsoft chat boot capable of conversation and of writing poems
- 139 of these poems were published

Creativity

- Poems written with ALGOL words (Arnaud 1965 https://www.codeshow.it/Attori/Arnaud)
- Perl Haiku (Wall, Hopskins, 1995 https://www.codeshow.it/Codici/Haiku)
- Esoteric languages https://www.codeshow.it/Linguaggi/Linguaggi_esoterici

Esolangs/1: Brainf*ck

```
 ++++++++++[>+++++++>++++++++++>++
+<<<-]>++.>+.+++++++

 ..+++.>++.<<+++++++++++++++.>.++
+.------.--------.>+.
```

Esolangs/2: Piet

Esolangs / 3: new ?

- A language composed only with notes and signs, like: 𝄆 ♩ ♪ ♫ ♬ ♯ ♭ ¼ 𝄽 𝄾 𝄿 𝄞 𝄢 𝄴 𝄇
- A neapolitan (…) language
- ...

Part III

Why learn to program?

- When teaching programming, often a teacher is blind: since s/he love programming, also students will love it
- But:
  - students don't start learning programming for love of computational thinking
  - students should find a personal motivation
- Fun is a great motivation; but programming is difficult and fun could soon disappear leaving place to frustration and anger
- Find a job is a goal for young adults, but not for children
- ...
- So?

Coding ideas

- Source codes are types of texts
- Programming (= coding) is a sub-type of writing
- Students could learn to use programming language as a way to express ideas
- Not in same sense in which one write poems or stories, but in the sense in which one write scenarios or music
- These are special types of text: they are active (or performative) text, they do things when read by computers

Running models

- Programming is a way to imagine and build a *model* (a running simulation) of a portion of the universe
- The reason why we build a digital simulation is that a model is far more cheap, simpler, faster than reality
- While running (and debugging…) the model, we understand something more about the universe we are simulating
- We use programming to test an hypothesis
  - we have an initial state
  - we build some rules
  - we put in data
  - we stand to see if our rules are powerful enough to represent the final state

Duties and exercises

- *Students* often do exercises only to get grades
- They don't think of exercises as a way to better learn theories by practice
- *Teachers* often assign duties only to assess students, to give grades
- They don't think of exercises as a way to help student to understand how to apply in the real case what has been told in theory

CS exercises

- In the same years in which computers were used to teach, they were also used to assess students
- Still now, computer assessment is seen as an advantage over human assessment:
  - "computer cannot be wrong"
  - they are good at crunching numbers
  - they don't get tired after one thousand of exercises
- But this is a really poor way to use computer

Authentic assessment

- Authentic assessment is a vision of assessment in which exercises are as similar as possible to real problems
- On one hand we have classical closed quiz, with textual questions and a limited number of possible answers, that are supposed to verify the knowledge of the theory by the student
- On the other hand, there is the real world, that is more complex than theoretical world: there is noise, friction, dust, interaction.
- *Simulated worlds* are somewhere in the middle: we can build them (nearly) as complicated as we want.
- Inside a simulated world we can to authentic assessment

Meaningful exercises

- CS offers a wonderful way to test a theory
- "Imagine a robot capable of discussing with you" (Eliza, Weizenbaum)
- A real robot is a quite complex thing; but a model of a robot could be just a software writing sentences
- Building such a software leads to face difficulties:
  - limited resources (memory, processors, channels)
  - limit of the language (expressiveness, available data structures, ...)
  - dirty data
  - …
- *Deal with limits* is one of the original meaning of "computational thinking" as different from "logic thinking" (Jeannette Wing)

Open artifacts

- A good way to teach programming is perhaps to stay in the middle:
  - use open environments (educational languages)
  - but start with something already (partially) running
- Give a stub, a sketch with ideas, expected output, examples
- But also with some functions, or libraries, to start with

Learning Units

- A learning unit is not a final piece of software, ready, that the student should use
- It should be a mini-environment in which the student can enter, starting to see and understand, but that requires an effort to be really useful and usable
- Programming should be presented as a way to model the world (physical, biological, linguistic, historical, geographical, mathematical, … world)
- There should be a goal: something unclear to better understand, something uncertain, or too complex to be understood without a dynamical representation

The LU Project

- Who:
  - the age and class the LU is aimed to
- Why:
  - the learning objectives, pre-requisites and motivations respect to the interdisciplinary topic (Math, Physics, Music, History, Geography, ...)
  - the learning objectives, pre-requisites and motivations respect to Programming
- What and when:
  - learning materials you give to the students (files and/or pre-programmed functions)
  - a description of how the LU will be delivered in class/lab
- How:
  - an evaluation grid explaining how the characteristics of project made by the students contribute to the assessment (sufficient/good/outstanding)

https://twiki.di.uniroma1.it/twiki/view/CSeduA/WebHome